Boulder, Colorado, March 2001. LA-UR 01-1039

Why TCP Will Not Scale for the Next-Generation Internet

Eric Weigle[†], Wu-chun Feng[†‡], and Mark Gardner[†]
{ehw, feng, mkg}@lanl.gov

[†] Research & Development in Advanced Network Technology, Los Alamos National Laboratory, Los Alamos, NM 87545

[‡] Department of Computer & Information Science, Ohio State University, Columbus, OH 43210
School of Electrical & Computer Engineering, Purdue University, W. Lafayette, IN 47907

Until recently, the average desktop computer has been powerful enough to saturate any available network technology; but this situation is rapidly changing with the advent of Gigabit Ethernet (GigE) and similar technologies. While CPU speeds have improved by over 50% per year since the mid-1980s (or roughly doubling every 1.6 years) [3], network speeds have improved by nearly 100% per year from 10-Mb/s Ethernet in 1988 to 6.4 Gb/s HiPPI-6400/GSN [10] in 1998! Network speeds have finally surpassed the ability of the computer to fill the network pipe, and this situation will get dramatically worse due to the above trends as well as slowly increasing I/O bus speeds. While the average I/O bandwidth of a PC is expected to increase from 1.056 Gb/s (32-bit, 33-MHz PCI bus) to 4.224 Gb/s (64-bit, 66-MHz PCI bus) over the next 12-18 months, the widespread availability of HiPPI-6400/GSN (6.4 Gb/s) [10] this year and 10GigE (10 Gb/s) [6] next year will far outstrip the ability of a computer to fill the network.

Our experiments already demonstrate that a PC can no longer fully utilize available network bandwidth. With the default Red Hat Linux 6.2 OS running on dual processor 400-MHz PCs with Alteon AceNIC GigE cards on a 32-bit, 33-MHz PCI bus, the peak bandwidth achieved by TCP is only 335 Mb/s. With an 83% increase in CPU speed to 733 MHz, the peak bandwidth only increases by 25% to 420 Mb/s. These bandwidth numbers can be improved by about 5% by increasing the default send/receive buffer sizes from 64 KB to 512 KB, by another 10% by using interrupt coalescing, and by slightly more with further enhancements to the system set-up as described below. Unfortunately, TCP still utilizes only half the available bandwidth in the best case between two machines.

This implies that file or web servers wishing to fully utilize their available gigabit-per-second bandwidth will have trouble doing so. Furthermore, remote visualization projects of large data sets will be bound by TCP/IP stack performance, as described below.

To create an environment that delivers maximum bandwidth, we removed all processes from the Red Hat Linux 6.2 OS except for the kernel, init, and a shell, turned off virtual memory and interrupt coalescing, and set the benchmarking software as a real-time process. We then configured a single machine to run the tests over its loopback interface (thus not touching the PCI bus or network at all). In this best case, we found that the machine spends a majority of time in the protocol stack rather than transmission into the network and results in only 485 Mb/s on the 400-MHz machine and 590 Mb/s on the 733-MHz machine. Despite this optimal (albeit unrealistic) environment, these numbers demonstrate that it is impossible for TCP running with Linux on this hardware to achieve gigabit speeds between machines. Thus, even if the hardware-based bottleneck at the host interface is alleviated, e.g., the future InfiniBand [1], the software-based bottleneck at the host interface, e.g., TCP/IP, will persist.

Additional problems that work to sabotage achieving high bandwidth include TCP's flow- and congestion-control mechanisms and the use of 1500-byte maximum transmission units (MTUs) in Ethernet. For example, TCP Reno generally viewed as the ubiquitous TCP has been shown to induce bursty and chaotic behavior in network traffic [5, 7, 9, 2], and its congestion-control mechanism never uses more than 75% of the available link bandwidth on average [11], e.g., under 750Mbps on a gigabit Ethernet link. Other versions of TCP begin to address this problem, but it is still an open issue. In addition, with TCP's default flow-control window, connections with large bandwidth-delay products cannot keep the network pipe full and suffer significant performance degradation when compared to the performance on LANs. Finally, because of Ethernet's 1500-byte MTU (note: a few non-standard implementations support ”jumbo packets” at 9000 bytes), a fully utilized 10-Gb/s Ethernet line would produce a minimum of 830,000 packets per second. If the network interface card (NIC) does not provide interrupt coalescing, each packet would interrupt the host and cause a jump to the interrupt handler. As 2000 cycles per interrupt is the minimum for Linux 2.2.17 to switch from a user process to the interrupt handler and back, a 1.6-GHz CPU would be required merely to handle the interrupt *without* doing any other processing such as receiving a packet!

While jumbo packets and interrupt coalescing may improve delivered bandwidth, there are problems with both. Jumbo packets can only effectively be used in a local-area network; wide-area networks will likely fragment them to 1500 bytes or drop them entirely if we set the ”don't fragment” bit, which we would want to do to avoid the cost of fragmentation and reassembly in high-performance systems. Furthermore, in the worst case, a packet may cross an old link that only guarantees the Internet standard 576-byte transmission unit [8]. Jumbo packets also induce blocking on networks that do not allow out-of-band data (such as Ethernet), where a small high priority packet from one connection may be enqueued in a switch behind several large jumbo packets from another connection and have to wait. Switched networks that either have a smaller MTU (e.g. ATM at 53 bytes) or support out-of-band data (e.g. Myrinet) do not suffer from this problem.

Interrupt coalescing forces us to make a tradeoff between bandwidth and latency. That is, by amortizing the cost of an interrupt over several packets, we can increase delivered bandwidth but the latency increases for each individual packet. Other related approaches which offload work to the NIC are not scalable. For example, some programmers have tried to place the user's network buffers on the network interface card (thus decreasing the total number of copies end to end). These approaches require more complex and expensive cards (surprisingly, even a few megabytes of memory will account for a large percent of the cost of a NIC) and the approach does not scale as the number of connections increases— servers may handle thousands of concurrent connections; even with only the average 64KB TCP buffer, that would require much more memory than the 4 or 8MB of memory than is commonly placed on high-speed NICs today. Seen another way, a 1 Gigabit connection across the United States will have a delay of at least 100ms; this gives a bandwidth-delay product of 100Mb or 12.5MB per connection. Even a single connection of this type will not fit on an 8MB NIC.

Moreover, even when the above technical issues are resolved, we will still have major security implications from having fast, always-on connections. When delivered bandwidth is constrained by host speeds instead of the link speeds, it becomes trivial to perform denial-of-service attacks. A few PCs have comparable power to the average web server; and if they produce requests as fast as possible, and the data stream is not throttled by the link speed, all requests would arrive and likely overwhelm the server.

Two solutions have been proposed in the literature. Unfortunately, neither solution completely solves the problems. First is the argument that we should simply use a reliable UDP over networks with large bandwidth-delay products. This may work in local-area networks without competing flows, but it is *not* a solution as it would lead to congestion collapse if widely used, as evidenced by the congestion collapse of the mid-1980's [4]. The right solution is to fix TCP, as it is the most widely used transport protocol and contains congestion-control mechanisms. Second is the proposed use of smart queueing to punish non-conformant flows, e.g., UDP streams which do not respond to loss or streams which appear to be using more than their fair share of bandwidth (such as those involved in a denial of service attack). This approach will also not scale— it generally requires state to be kept on a per flow basis, and even at only a few bytes per connection a backbone router handling millions of connections will be unable to do so economically. Furthermore, attackers would merely have to forge IP headers or stripe packets across multiple connections to circumvent this approach. Control measures that only work if the enemy cooperates are worthless.

In summary, this research brings up a number of issues that need to be addressed as we head into the Next-Generation Internet era:

1. host-interface bottlenecks (hardware and software)
2. protocol off-loading to the network interface card
3. next-generation flow-control and congestion-control mechanisms
4. high-performance networking in a secure environment
5. smart routers to punish non-conforming flows, e.g., UDP streams or streams from hosts generating distributed denial-of-service attacks

References

[1] D. Cassiday. InfiniBand Architecture Tutorial. Hot Chips 12 Tutorial, August 2000.

[2] W. Feng and P. Tinnakornsrisuphap. The Failure of TCP in High-Performance Computational Grids. In *Proceedings of SC 2000*, November 2000.

[3] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach (2nd edition)*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[4] V. Jacobson. Congestion Avoidance and Control. *ACM Computer Communications Review*, 18(4):314–329, August 1988.

[5] W. Leland, M. Taqqu, W. Willinger, , and D. Wilson. On the Self-Similar Nature of Network Traffic (Extended Version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[6] Nortel Networks. 10 Gigabit Ethernet: Unifying the LAN, MAN, and WAN. White Paper, April 2000.

[7] V. Paxson and S. Floyd. Wide-Area Traffic: The Failure of Poisson Modeling. *ACM SigComm 1994*, pages 257–268, 1994. ftp://ftp.ee.lbl.gov/papers/poisson.ps.Z.

[8] J. Postel. The TCP Maximum Segment Size and Related Topics (RFC879), November 1983.

[9] P. Tinnakornsrisuphap, W. Feng, , and I. Philp. On the Burstiness of the TCP Congestion-Control Mechanism in a Distributed Computing System. *Proceedings of the 20th International Conference on Distributed Computing Systems (ICDCS'00)*, April 2000.

[10] D. Tolmie, T. M. Boorman, A. DuBois, D. DuBois, W. Feng, , and I. Philp. From HiPPI-800 to HiPPI-6400: A Changing of the Guard and Gateway to the Future. In *Proceedings of the 6th International Conference on Parallel Interconnects (PI'99)*, October 1999.

[11] A. Veres and M. Boda. The Chaotic Nature of TCP Congestion Control. In *Proceedings of IEEE Infocom 2000*, March 2000.